



AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa inżynierska

*Implementacja systemu uwierzytelniania z zastosowaniem
Negatywnych Baz Danych*

Implementation of authentication system using Negative Databases

Autor:	Grzegorz Nieużyła
Kierunek studiów:	Informatyka
Opiekun pracy:	dr inż. Piotr Szwed

Kraków, 2020

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Spis treści

1. Wprowadzenie	7
1.1. Cele pracy	7
1.2. Zawartość pracy	8
2. Solwery SAT	9
2.1. Problem spełnialności	9
2.2. Opis działania solwerów SAT	10
2.2.1. Solwery kompletne	10
2.2.2. Solwery niekompletne	12
3. Negatywne Bazy Danych - opis teoretyczny	13
3.1. Opis działania	13
3.2. Zastosowanie w systemach uwierzytelniania	14
3.3. Algorytmy generacji Negatywnych Baz Danych	14
3.3.1. Algorytm prefiksowy	14
3.3.2. Algorytm <i>Randomize_NDB</i>	16
3.3.3. Algorytm <i>0-Hidden</i>	20
3.3.4. Algorytmy <i>1-Hidden</i> i <i>2-Hidden</i>	20
3.3.5. Algorytm <i>Q-Hidden</i>	21
3.3.6. Algorytm <i>K-Hidden</i>	23
3.3.7. Algorytm hybrydowy	24
3.4. Wykorzystywanie solwerów SAT w celu uzyskania przeciwobrazu Negatywnej Bazy Danych	26
4. Testy algorytmów	29
4.1. Opis testowania za pomocą solwerów SAT	29
4.2. Testy metod weryfikujących poprawność rekordów pozytywnych	29
4.3. Testy algorytmów generacji <i>NDB</i>	29
5. Implementacja systemu uwierzytelniania	31
5.1. Reprezentacja danych	31

5.2.	Algorytm tworzenia użytkownika	31
5.3.	Algorytm uwierzytelniania użytkownika	31
5.4.	Działanie aplikacji	31
6.	Wnioski	33
6.1.	Bezpieczeństwo przedstawionej implementacji	33
6.2.	Możliwe rozszerzenia	33

1. Wprowadzenie

1.1. Cele pracy

Celem niniejszej pracy jest implementacja i przetestowanie systemu uwierzytelniania oferującego większe bezpieczeństwo niż standardowy schemat generowania skrótu hasła za pomocą funkcji generacji klucza (np. *PBKDF2*, *bcrypt*) i przechowywaniu w standardowej (pozytywnej) bazie danych.

Jak wynika z raportu Google[1], większość internautów ma problem z zachowaniem zasad bezpieczeństwa dotyczącego używania mediów społecznościowych i portali internetowych. 66% procent badanych przyznało się do używania jednego hasła do różnych kont. W przypadku wykradzenia bazy danych pojedynczego portalu i uzyskaniu postaci jawnych haseł istnieje ryzyko uzyskania dostępu również do innych aplikacji przez atakującego. W takiej sytuacji wyciek danych z amatorskiego forum tematycznego może zagrozić naszemu kontu bankowemu lub poczcie elektronicznej.

Główną przyczyną nadużywania jednego hasła jest konieczność zapamiętania danych logowaniu do wielu stron internetowych – 75% internautów ma problem z ich zarządzaniem.

Istnieje szereg metod, które indywidualny użytkownik może zastosować w celu polepszenia swojego bezpieczeństwa w Internecie, ale nie cieszą się one dużą popularnością. Według raportu tylko 37% internautów używa uwierzytelnienia wielopoziomowego, a menadżerów haseł umożliwiających stosowanie różnych i silnych danych logowania – 15%.

Częściowym rozwiązaniem tego problemu może być wprowadzenie dodatkowych warstw bezpieczeństwa bezpośrednio w systemach uwierzytelniania w celu utrudnienia uzyskania nieautoryzowanego dostępu do wrażliwych danych. Takie podejście pozwoli podnieść bezpieczeństwo wszystkich użytkowników, niezależnie od indywidualnych preferencji i nawyków.

Założeniem systemu jest zamienienie reprezentacji w sposób jawny na Negatywną Bazę Danych (*NDB*) co pozwoli dodać dodatkową warstwę bezpieczeństwa która znacząco utrudni uzyskanie haseł użytkownika w przypadku kradzieży bazy danych.

W tym celu opisałem różne algorytmy prezentowane w dostępnej literaturze, przedstawiłem schemat ich działania i porównałem je pod względem bezpieczeństwa.

Rezultatem wyjściowym algorytmów generacji *NDB* jest zbiór ciągów tekstowych, które można jednoznacznie sprowadzić do zbioru formuł logicznych CNF, dlatego przeprowadzone zostały testy z wykorzystaniem solverów SAT mające na celu zasymulowanie ataku na powstałą *NDB*.

Koncept negatywnej reprezentacji danych wywodzi się z immunologii. Badania wykazały, że limfocyty T wykorzystują podobny proces w celu rozróżnienia między sobą i innymi komórkami [2].

1.2. Zawartość pracy

2. Solwery SAT

2.1. Problem spełnialności

Problem spełnialności (ang. *Boolean satisfiability problem*, w skrócie SAT) polega na sprawdzeniu, czy dana formuła logiczna posiada odpowiednie przypisanie zmiennych przy których całość wyrażenia będzie ewaluowana do prawdy.

Wyrażenia wykorzystywane w problemie SAT składają się z:

- zmiennych - $x_1, x_2 \dots x_n$, mogących przybrać wartość prawdziwą (*true*, T , \top , 1) lub fałszywą (*false*, F , \perp , 0)
- operatora koniunkcji - *AND*, \wedge
- operatora alternatywy - *OR*, \vee
- operatora negacji - *NOT*, \neg
- nawiasów - $()$

Formuły SAT przedstawia się w postaci CNF (koniunkcyjna postać normalna, ang. *conjunctive normal form*) której struktura jest następująca: formuła jest koniunkcją klauzul, a klauzula jest alternatywą literałów (tj. zmiennych lub ich negacji):

$$(x_{11} \vee x_{12} \vee \dots \vee x_{1n}) \wedge (x_{21} \vee x_{22} \vee \dots \vee x_{2n}) \wedge \dots \wedge (x_{m1} \vee x_{m2} \vee \dots \vee x_{mn})$$

Naiwny algorytm SAT jest trywialny - wystarczy dla każdej możliwej kombinacji zmiennych $x_1 \dots x_n$ sprawdzić czy formuła jest spełnialna. Sprawdzenie następuje w czasie liniowym od wartości n , zatem cały algorytm ma złożoność obliczeniową $O(2^n)$ - nie do przyjęcia w praktycznie żadnych zastosowaniach poza bardzo małymi wartościami n .

Problem SAT jest pierwszym problemem dla którego udowodniono że jest *NP*-zupełny (Twierdzenie Cooka-Levina [11, 12]). Oznacza to, że można w czasie wielomianowych sprowadzić każdy problem decyzyjny zawierający się w *NP* do SAT.

Problem należący do *NP* charakteryzuje się tym, że zweryfikowanie pojedynczego rozwiązania jest możliwe w czasie wielomianowym dla deterministycznej Maszyny Turinga, oraz że sprawdzenie wszystkich możliwości (a więc rozstrzygnięcie problemu) jest możliwe również w czasie wielomianowym, ale

dla niedeterministycznej Maszyny Turinga - przez jednoczesne sprawdzenie każdej kombinacji. Wynika z tego również, że $P \subseteq NP$.

Nie każda możliwa formuła CNF stanowi problem NP -zupełny. Twierdzenie o dychotomii Schaefera określa szczególne instancje problemu należące do P (przy założeniu, że $P \neq NP$) [13]:

1. Formuła jest spełnialna jeśli wszystkie zmienne przyjmują wartość 0
2. Formuła jest spełnialna jeśli wszystkie zmienne przyjmują wartość 1
3. Każda klauzula ma co najwyżej jeden literał pozytywny (każda klauzula jest klauzulą Horna)
4. Każda klauzula ma co najwyżej jeden literał negatywny (każda klauzula jest dualną klauzulą Horna)
5. Każda klauzula ma co najwyżej dwa literały (2-CNF)
6. Formuła jest równoznaczna systemowi równań linowych $x_1 \oplus x_2 \oplus \dots \oplus x_n = c$, gdzie $x_i, c \in \{0, 1\}$ a $\oplus \equiv$ dodawanie mod_2

2.2. Opis działania solwerów SAT

Solwery SAT można podzielić na dwie kategorie – kompletne i niekompletne. Algorytmy kompletne gwarantują, że jeśli istnieje spełniające przypisanie dla analizowanej formuły to zostanie zwrócony wynik pozytywny. Natomiast wynik działania solwera niekompletnego może być albo pozytywny – istnieje odpowiednie przypisanie, albo niezdefiniowany – algorytm nie znalazł rozwiązania w określonej liczbie prób.

2.2.1. Solwery kompletne

Algorytmy kompletne do rozstrzygania problemów SAT korzystają z szeregu metod do których należą: kwantyfikacja egzystencjalna, wnioskowanie, przeszukiwanie oraz wnioskowanie i przeszukiwanie [14].

2.2.1.1. Kwantyfikacja egzystencjalna

Kwantyfikacja egzystencjalna jest klasą algorytmów upraszczających zadaną formułę CNF do postaci spełnialnej $\{\}$ lub niespełnialnej $\{\emptyset\}$. W tym celu definiuje się operator \exists zmiennej P w formule Δ :

$$\exists P \Delta \stackrel{\text{def}}{=} (\Delta|P) \vee (\Delta|\neg P)$$

gdzie operator $'|'$ jest *warunkowaniem* i definiuje się je następująco:

$$\Delta|P = \{\alpha - \{\neg P\} \mid \alpha \in \Delta, P \notin \alpha\} \quad [14]$$

Proces wnioskowania można intuicyjnie przedstawić jako ustawienie zmiennej P na wartość prawdziwą (lub fałszywą w przypadku $\neg P$) i zredukowanie zbioru klauzul usuwając te, które są spełnione oraz usuwając z pozostałych literały mające wartość fałszywą. Kwantyfikator $\exists P \Delta$ określa formułę po wyeliminowaniu zmiennej P przez rozpatrzenie obu możliwych wartości.

Z punktu widzenia problemu spełnialności najważniejszą właściwością jest to, że formuła Δ jest spełnialna wtedy i tylko wtedy, gdy $\exists P \Delta$ jest spełnialna. Pozwala to na stopniowe eliminowanie zmiennych aż do osiągnięcia pustej klauzuli lub pustej formuły. Kwantyfikacja egzystencjalna jest wykorzystywana m.in. w algorytmie DP oraz w symbolicznym rozwiązywaniu SAT.

2.2.1.2. Wnioskowanie

Upraszczenie formuł przez wnioskowanie opiera się na usuwaniu redundantnych literałów tj. wykluczających się nawzajem. Przykładowo w formule:

$$\{\{x_1, x_2, \neg x_3\}, \{\neg x_2, x_4, x_5\}\}$$

zmienna x_2 występuje w postaci zanegowanej i niezaniegowanej w różnych klauzulach. Oznacza to, żeby cała formuła była spełniona, co najmniej jeden pozostały literał musi się ewaluować do prawdy. Można zatem zastosować następujące przekształcenie:

$$\{\{x_1, \neg x_3, x_4, x_5\}\}$$

Dana formuła jest niespełnialna wtedy i tylko wtedy, gdy istnieje wnioskowanie prowadzące do pustej klauzuli, zatem po wyczerpaniu wszystkich możliwości wnioskowania możemy ustalić, że problem jest spełnialny.

Do algorytmów wykorzystujących tę metodę należą algorytm Stålmarcka i HeerHugo.

2.2.1.3. Przeszukiwanie

Problem SAT można przedstawić jako drzewo decyzyjne przypisania zmiennych, gdzie liście są wszystkimi możliwymi kombinacjami literałów. Metoda przeszukiwania polega na znalezieniu liścia spełniającego formułę. Używa się do tego algorytmu przeszukiwania wszerz oraz warunkowania w celu ograniczenia drzewa.

Dodatkowo używa się także *propagacji jednostkowej* (ang. *unit propagation*) za pomocą której można odrzucić lub potwierdzić poprawność danego poddrzewa we wczesnej fazie analizy przez spełnienie klauzul jednostkowych (posiadających tylko jeden literał).

Wzorcowym zastosowaniem tej metody jest algorytm *DPLL*.

2.2.1.4. Wnioskowanie i przeszukiwanie

Wadą podejścia zastosowanego w *DPLL* jest fakt, że podczas cofania się po drzewie w kierunku korzenia nie bierze się pod uwagę zbioru zmiennych, które spowodowały odrzucenie danego poddrzewa. Dlatego w tej metodzie stosuje się tzw. niechronologiczne cofanie – podczas konfliktu literałów wyznacza się zbiór powodujących go zmiennych i zmienia się ich wartości, pomijając te nie biorące udziału w konflikcie.

Dodatkowo podczas działania algorytmu wprowadza się dodatkowe klauzule przedstawiające zależności, które pozwalają wykrywać nieścisłości bliżej korzenia, jeszcze bardziej skracając czas działania.

Do tej klasy należy większość współczesnych, najbardziej optymalnych algorytmów kompletnych rozstrzygania problemu SAT – m.in. *zChaff*, *MiniSAT*, *PicoSAT* i *Siege*.

2.2.2. Solwery niekompletne

Solwery nie gwarantujące znalezienia rozwiązania opierają się na *stochastycznym przeszukiwaniu lokalnym*. Takie rozwiązanie pozwala na znaczne mniejsze czasy wykonywania dla odpowiednich formuł niż algorytmy wykorzystujące DPLL, ale w przypadku trudnych problemów zamiast zdegenerowania złożoności obliczeniowej do $O(2^n)$ zostaje zwrócony wynik nieokreślony.

Czołowymi solwerami z tej kategorii są *GSAT* i *WalkSAT* [15].

2.2.2.1. GSAT

Schemat działania *GSAT* polega na wygenerowaniu losowego przypisania i zachłannym *odwracaniu* (ang. *flipping*) wartości zmiennych w celu zminimalizowania liczby niespełnionych klauzul. Ilość odwróceń oraz liczbę prób z wygenerowaniem nowego ciągu początkowego można modyfikować za pomocą odpowiednich parametrów.

2.2.2.2. WalkSAT

WalkSAT rozszerza algorytm stopniowego odwracania zmiennych używany w *GSAT*, redukując stopień zachłanności. Na początku każdej iteracji wybierana jest losowa klauzula, i w niej szukana jest zmienna, której zanegowanie nie zmieni żadnej innej klauzuli na niespełnioną. Jeśli nie została znaleziona, *WalkSAT* wybiera z danej klauzuli losową zmienną lub taką, której zmiana spowoduje spełnienie jak największej ilości pozostałych klauzul. Wprowadzony jest dodatkowo parametr szumu $p \in [0, 1]$ określający jak często wybierane jest podejście nie-zachłanne. Dla losowych formuł 3SAT optymalna wartość szumu wynosi $p = 0.57$ [15].

3. Negatywne Bazy Danych - opis teoretyczny

3.1. Opis działania

Główną operacją wykonywalną na *NDB* jest sprawdzenie czy dany rekord znajduje się w bazie. Przyjmując U jako oznaczenie uniwersum języka binarnego o długości l a DB jako zbiór wszystkich rekordów, każdy o długości l , *NDB* przechowuje zbiór $U - DB$ [3]. Takie dane są niepraktycznie do zareprezentowania w postaci nieskompresowanej z uwagi na wielkość, dlatego stosuje się wyrazy nad alfabetem $\{0, 1, *\}$ gdzie symbol $*$ może oznaczać zarówno 0 lub 1 w jawnej reprezentacji bitowej. Pozycje na których znajduje się wartość 0 lub 1 są *ustalone* a z wartością $*$ - *nieustalone*.

Każdy taki wyraz odpowiada jednemu lub wielu elementom $U - DB$ i jest sprowadzany do formuły logicznej (Tabela 3.1). Z założenia algorytm sprawdzający przynależność do DB sprawdza czy jakakolwiek formuła z *NDB* jest spełniana przez dany rekord. Dane znajdują się w DB wtedy i tylko wtedy gdy żadna formuła nie zostanie spełniona.

Taki model działania wymusza na danych stałą wielkość, co jednak nie stanowi problemu w przypadku przechowywania skrótów haseł które mają stałą, zależną od konkretnego algorytmu długość. Dla danych o zmiennych rozmiarach (np. nazwy użytkownika) można zastosować funkcję hashującą lub algorytmy zwiększające długość ciągu bitowego do stałej wartości typu PKCS#5 lub PKCS#7. Należy jednak pamiętać, że zwiększenie długości rekordu znacznie wydłuża czas generacji bazy oraz zajmowaną pamięć dla niektórych algorytmów.

rekord NDB	formuła logiczna
011*	$\neg x_1 \wedge x_2 \wedge x_3$
001*	$\neg x_1 \wedge \neg x_2 \wedge x_3$
1*1*	$x_1 \wedge x_3$
0*0*	$\neg x_1 \wedge \neg x_3$
1*00	$x_1 \wedge \neg x_3 \wedge \neg x_4$

Tabela 3.1. Reprezentacja formuł logicznych za pomocą NDB kodującej zbiór $DB = \{1001, 1101, 1110\}$

3.2. Zastosowanie w systemach uwierzytelniania

NDB może być wykorzystana w każdym systemie, gdzie podstawową operacją na danych jest sprawdzenie czy dany rekord znajduje się w bazie. Jednym z najpopularniejszych systemów uwierzytelniania jest metoda oparta na loginie i hasle. Użytkownik danej aplikacji przy zakładaniu konta podaje hasło, które następnie warstwa serwerowa danej aplikacji przechowuje jako wynik nieodwracalnej funkcji hashującej.

W przypadku nieautoryzowanego dostępu do bazy danych i używanego algorytmu uzyskiwania skrótu z hasła, atakujący może uzyskać wartość pierwotną mało skomplikowanych haseł za pomocą np. metody słownikowej. Modyfikując powyższy algorytm składując skróty jako rekordy w NDB uniemożliwiamy łatwą iterację wszystkich danych, jednocześnie pozostawiając łatwy dostęp do informacji czy użytkownik o podanym loginie i hasle ma dostęp do aplikacji.

3.3. Algorytmy generacji Negatywnych Baz Danych

3.3.1. Algorytm prefiksowy

Najprostszym ze sposobów generowania Negatywnych Baz Danych jest zaproponowany przez Fernando Esponda algorytm prefiksowy [3, 4]. Został on opracowany w celu udowodnienia że proces generowania NDB z rekordów DB jest możliwy w rozsądnej złożoności czasowej i pamięciowej.

Algorytm 1: Algorytm prefiksowy

Dane: DB - zbiór rekordów do zarepresentowania w NDB, l - długość rekordu DB

Wynik: Zbiór rekordów NDB

```

1  $Prefix_n(V)$  - Prefiks n-znakowy rekordu  $V$ 
2  $len(V)$  - Długość rekordu  $V$ 
3  $W_i = \{\}$ ;
4  $i = 0$ ;
5 while  $i < l$  do
6    $W_{i+1} =$  Zbiór wszystkich  $i + 1$ -znakowych ciągów bitowych  $V_p$  nie będących prefiksem
      żadnego rekordu  $DB$  i dla których  $Prefix_i(V_p) \in W_i$ 
7   foreach  $V_p$  in  $W_{i+1}$  do
8     Stwórz rekord NDB o długości  $l$  którego  $V_p$  jest prefiksem a na pozostałych pozycjach
      jest symbol  $*$  i dodaj do zbioru wyjściowego NDB
9   end
10   $i = i + 1$ ;
11   $W_i =$  Zbiór wszystkich  $i$ -znakowych prefiksów rekordów  $DB$ 
12 end
```

Powyższa metoda polega na generowaniu coraz dłuższych prefiksów które nie pokrywają się ze zbiorem DB . W ten sposób na początku tworzone są rekordy odpowiadające znacznej części $U - DB$. Czasami występuje potrzeba zdefiniowania pewnych rekordów explicite bez wykorzystania symbolu $*$ jeżeli każdy możliwy prefiks jest także prefiksem rekordu z DB . Przykładowy wynik działania znajduje się w tabeli 3.2.

DB	U - DB	NDB
0000	0001	10**
0110	0011	010*
0010	0100	111*
1101	0101	0001
	0111	0011
	1000	0111
	1001	1100
	1010	
	1011	
	1100	
	1110	
	1111	

Tabela 3.2. Rezultat działania algorytmu prefiksowego

Algorytm prefiksowy jest deterministyczny i każdy powstały rekord reprezentuje unikalną, nie pokrywającą się część $U - DB$ [3]. Powoduje to, że algorytm uzyskiwania zbioru DB z otrzymanej NDB

nie wymaga sprowadzenia do problemu SAT. Wystarczy jedynie odpowiednio posortować rekordy i wyznaczyć przedziały pomiędzy nimi, co pokazują w algorytmie 2.

Algorytm 2: Algorytm odwracający *NDB* wygenerowaną za pomocą algorytmu prefiksowego

Dane: *NDB* - zbiór rekordów do odwrócenia, l - długość rekordu *NDB*

Wynik: Zbiór rekordów DB

- 1 Posortuj zbiór *NDB*
 - 2 V_l = rekord *DB* wypełniony symbolami '0' o długości l
 - 3 **foreach** V **in** *NDB* **do**
 - 4 $V_h = V$ z zamienionymi symbolami '*' na '0'
 - 5 $R = \text{Rekordy_DB_W_Przedziale}(V_l, V_h)$
 - 6 Dodaj R do zbioru wynikowego
 - 7 $V_l = V$ z zamienionymi symbolami '*' na '1'
 - 8 **end**
 - 9 V_h = rekord *DB* wypełniony symbolami '1' o długości l
 - 10 $R = \text{Rekordy_DB_W_Przedziale}(V_l, V_h)$
 - 11 Dodaj R do zbioru wynikowego
-

Algorytm 3: *Rekordy_DB_W_Przedziale*

Dane: V_1, V_2 - rekordy *DB*

Wynik: Zbiór rekordów DB w przedziale (V_1, V_2)

- 1 Dodaj binarnie 1 do V_1
 - 2 **while** $V_1 \neq V_2$ **do**
 - 3 Dodaj V_1 do zbioru wynikowego.
 - 4 Dodaj binarnie 1 do V_1
 - 5 **end**
-

Czas wykonywania procedury generacyjnej wynosi $O(l|DB|)$, jednak w przypadku zapisywania wyniku do bazy wzrasta do $O(l^2|DB|)$ gdyż konieczna jest serializacja każdego rekordu. Złożoność pamięciowa dla optymalnej implementacji wynosi $O(l|DB|)$ w przypadku gdy poprzednio generowane rekordy *NDB* nie są przetrzymywane w pamięci. Dla danego zbioru *DB* generowane jest $O(l|DB|)$ rekordów co sprowadza się do wielkości powstałej bazy danych wynoszącej $O(l^2|DB|)$.

Przestawiony powyżej algorytm odwracający (2) wykonuje się w czasie $O(|NDB|) = O(l|DB|)$ co jest wynikiem lepszym od procedury generacyjnej z zapisem do bazy.

3.3.2. Algorytm *Randomize_NDB*

Algorytm prefiksowy generuje poprawne rekordy *NDB*, jednak nie jest praktyczny w żadnych zastosowaniach związanych z bezpieczeństwem, ponieważ wynik jego działania jest stosunkowo prosto

sprowadzić do postaci pozytywnej. Aby temu zaradzić, Fernando Esponda w swojej pracy[3] zaproponował niedeterministyczny algorytm mający na celu wprowadzić rekordy które nie odpowiadają jedynie prefiksom elementów z $U - DB$ i są trudniejsze do odwrócenia.

Algorytm 4: Algorytm *Randomize_NDB*

Dane: DB - zbiór rekordów do zarepresentowania w NDB, l - długość rekordu DB

Wynik: Zbiór rekordów NDB

```

1   $Prefix_n(V)$  - Prefiks  $n$ -znakowy rekordu  $V$ 
2   $len(V)$  - Długość rekordu  $V$ 
3   $\pi$  = losowa permutacja o długości  $|V_{pe}|$ 
4   $W_i$  = zbiór wszystkich ciągów  $l$ -bitowych
5   $\pi(DB) \equiv \{\pi(V) \mid V \in DB\}$ 
6   $i = \lceil \log_2(l) \rceil$ ;
7  while  $i < l$  and  $W_i \neq \emptyset$  do
8       $W_{i+1}$  = Zbiór wszystkich  $i + 1$ -znakowych ciągów bitowych  $V_p$  nie będących prefiksem
          żadnego rekordu  $\pi(DB)$  i dla których  $Prefix_i(V_p) \in W_i$ 
9      foreach  $V_p$  in  $W_{i+1}$  do
10         Dopełnij  $V_p$  do długości  $l$  wstawiając znaki nieustalone '*' na końcu
11          $j$  = losowa liczba z przedziału  $[1, l]$ 
12         for  $k = 1$  to  $j$  do
13              $n$  = losowa liczba z przedziału  $[1, \log_2(l)]$ 
14              $P = n$  losowych nieustalonych pozycji z  $V_p$ 
15              $X$  = Zbiór rekordów powstałych przez zastąpienie pozycji  $\in P$  przez wszystkie
                możliwe kombinacje bitowe ( $2^n$  rekordów)
16             foreach  $V_q$  in  $X$  do
17                  $V_{pg} = \text{Pattern\_Generate}(\pi(DB), V_q)$ 
18                 Dodaj  $\pi^{-1}(V_{pg})$  do zbioru rekordów wyjściowych
19             end
20         end
21     end
22      $i = i + 1$ 
23      $W_i$  = Zbiór wszystkich  $i$ -znakowych prefiksów rekordów  $DB$ 
24 end

```

Algorytm ten działa na podobnej zasadzie co algorytm prefiksowy (rozdział 3.3.1) z pewnymi modyfikacjami. Na początku kolejność bitów w DB jest mieszana za pomocą losowej permutacji aby pozycje zdefiniowane w generowanej prefiksowej NDB nie były skumulowane na początku wyrazów. Następnie dla każdego powstałego negatywnego rekordu losuje się n pozycji nieustalonych i zastępuje

się go równoznacznym zbiorem rekordów które mają te pozycje ustalone. Powstałe ciągi są dodatkowo zaciemniane przez wstawienie na losowych pozycjach zamiast bitu zdefiniowanego znak $*$ zgodnie z algorytmem *Pattern_Generate*.

Wynik algorytmu jest niedeterministyczny co powoduje że dla takich samych zbiorów *DB* rezultat może się różnić. Generacja wielu redundantnych rekordów zwiększa odporność otrzymanej bazy na próby przywrócenia do postaci pozytywnej, jednak wiąże się to z ze zwiększeniem objętości NDB średnio $\frac{l^2}{2}$ razy w stosunku do algorytmu prefikсового.

Algorytm 5: Algorytm *Pattern_Generate*

Dane: *DB* - zbiór rekordów do zarepresentowania w NDB, V_q - rekord NDB do zaciemnienia

Wynik: Zaciemniony rekord NDB

```

1   $\pi$  = losowa permutacja o długości  $|V_{pe}|$ 
2   $SIV = \{ \}$  // wektor zmienionych bitów
3  for  $i = 1$  to  $|V_q|$  do
4       $\pi(V_q)' = \pi(V_q)$  z elementem o indeksie  $i$  zamienionym na symbol  $*$ 
5      if istnieje rekord w DB pokrywający się z  $\pi(V_q)'$  then
6          Dodaj  $i$  i bit o indeksie  $i$  do  $SIV$ 
7           $\pi(V_q) = \pi(V_q)'$ 
8      end
9  end
10  $t$  = losowa liczba z przedziału  $[0, |SIV|]$ 
11 if  $t > |SIV|$  then
12      $R = SIV$ 
13 else
14      $R = t$  losowych bitów z  $SIV$ 
15 end
16  $V_k = \pi(V_q)$ 
17 for indeks, bit in  $R$  do
18      $V_k[indeks] = bit$ 
19 end
20 Zwróć  $\pi^{-1}(V_k)$ 

```

Zmiany względem algorytmu prefiksowego oraz procedura *Pattern_Generate* gwarantują że żaden ze zmodyfikowanych rekordów nie będzie odpowiadał żadnemu elementowi *DB* oraz że wynik działania pozostanie kompletnym odzwierciedleniem $U - DB$, ponieważ każda modyfikacja rekordu rozszerza zakres pokrywanych ciągów wprowadzając redundantne informacje. W [3] przedstawiony jest dowód że problem rekonstrukcji *DB* z *NDB* jest NP-trudny (każdą instancję 3-SAT można sprowadzić do *NDB*).

Powyższy algorytm w teorii jest zdolny do tworzenia trudnych instancji, ale nie posiada żadnych mechanizmów umożliwiających ingerencję w jego działanie – wynik jest zawsze losowy. Prawdopodobieństwo uzyskania trudnej do złamania kombinacji rekordów jest bardzo małe, co powoduje że w znacznej większości przypadków wynikowa *NDB* może być odwrócona niemal natychmiast przez współczesne solwery SAT, co pokazują w dalszych rozdziałach.

DB	U - DB	NDB
0000	0001	*001
0110	0011	00*1
0010	0100	0011
1101	0101	010*
	0111	0*11
	1000	1**0
	1001	10**
	1010	101*
	1011	**11
	1100	1*1*
	1110	1100
	1111	111*

Tabela 3.3. Rezultat działania algorytmu *Randomize_NDB*

3.3.3. Algorytm 0-*Hidden*

Następujące algorytmy powstały przez zastosowanie procedur generowania trudnych instancji SAT. Jedną z nich jest **0-*Hidden*** służąca do generacji formuł 3-SAT nie posiadających rozwiązania, co pozwala na testowanie solverów pod względem wykrywania braku spełnialności [5].

Algorytm 6: Algorytm 0-*Hidden*

Dane: l - liczba zmiennych, r - współczynnik ilości klauzul

Wynik: Zbiór klauzul 3CNF

```

1  $n = l * r$ 
2  $W = \{ \}$ 
3 while  $|W| \neq n$  do
4     Wybierz 3 losowe zmienne
5     Stwórz klauzulę 3CNF używając wylosowanych zmiennych, z losowymi znakami
6     Dodaj klauzulę do zbioru wynikowego  $W$ 
7 end
```

Powyższy algorytm generuje formułę CNF, która z dużym prawdopodobieństwem nie jest spełnialna i jest trudna do rozwiązania przez solwery SAT [5, 6]. Modyfikując parametr r możemy wpłynąć na rozmiar formuły, wartość $r \approx 4.27$ jest wartością graniczną powyżej której problem prawie na pewno nie ma rozwiązania [5].

Z perspektywy Negatywnych Baz Danych można przyjąć że powstała formuła odpowiada zbiorowi $U - DB$, jeśli DB jest zbiorem pustym.

3.3.4. Algorytmy 1-*Hidden* i 2-*Hidden*

Algorytmy **1-*Hidden*** i **2-*Hidden*** są konstruowane podobnie jak **0-*Hidden***, z tą różnicą, że formuła wyjściowa ma odpowiednio co najmniej (i w znacznej większości dokładnie) jedno lub dwa rozwiązania.

W celu testowania możliwości solverów SAT co do znajdowania przypisania spełniającego daną formułę można generować losowe formuły (Algorytm 6) i odrzucać klauzule które przeczą ukrytemu rozwiązaniu A (**1-*Hidden***). Jednak spowoduje to, że rozkład losowy zostanie zaburzony i solwery mogą to „poczuć”, co doprowadzi je do ukrytego rozwiązania [6].

Aby temu przeciwdziałać można jednocześnie ukryć przypisania A oraz $\neg A$ (**2-*Hidden***), co spowoduje że algorytm przeszukujący będzie równoważnie „przyciągany” przez dwa przeciwne rozwiązania.

Powyższy schemat działania jest wykorzystany w procedurze **Q-*Hidden*** będącej rozszerzeniem tego konceptu.

3.3.5. Algorytm *Q-Hidden*

Metoda *Q-Hidden* do generacji trudnych formuł K-SAT zaproponowana w [5] rozszerza możliwości algorytmów z rozdziałów 3.3.3 i 3.3.4 przez dodanie parametru prawdopodobieństwa $q \in (0, 1)$. Każda wygenerowana klauzula składa się z k zmiennych których przypisanie pokrywa się z $t > 0$ literałów z prawdopodobieństwem q^t (gdzie t to numer kolejnego zgodnego literału). Oznacza to że im mniejsza wartość q tym bardziej formuła będzie wskazywać w przeciwnym kierunku niż ukryte rozwiązanie A .

Na podstawie tego schematu powstał algorytm mający na celu rozwiązanie problemów z algorytmami prefiksowym i Randomize_NDB tj. generowanie zbiorów rekordów które łatwo odwrócić za pomocą solverów SAT[7]

Algorytm 7: Algorytm *Q-Hidden*

Dane: s - ciąg bitowy, l - długość rekordu, r - wsp. ilości klauzul
 k - ilość ustalonych bitów w klauzuli, q - wsp. prawdopodobieństwa

Wynik: Zbiór rekordów NDB

```

1   $n = l * r$ 
2   $NDB = \{\}$ 
3  while  $|NDB| \neq n$  do
4       $\Upsilon = k$  różnych losowych pozycji w przedziale  $[1, l]$ 
5       $V =$  rekord  $NDB$  o długości  $l$  wypełniony znakami '*'
6      foreach  $i$  in  $\Upsilon$  do
7           $V[i] = s[i]$ 
8      end
9       $d = 0$ 
10     while  $d \neq 1$  do
11         foreach  $i$  in  $\Upsilon$  do
12              $p =$  losowa liczba rzeczywista z przedziału  $[0, 1]$ 
13             if  $q > p$  then
14                  $V[i] = \neg V[i]$ 
15                  $d = 1$ 
16             end
17         end
18     end
19     Dodaj rekord  $V$  do  $NDB$ 
20 end

```

W przeciwieństwie do poprzednich metod, za pomocą algorytmu *Q-Hidden* przy zastosowaniu odpowiednich parametrów q , r i k jest możliwe praktycznie stworzenie instancji NDB rozwiązywalnych

NDB		
*10*1	11**0	**100
11**0	**100	*000*
**100	*000*	0*01*
000	0*01*	*0*01
0*01*	*0*01	01*1*
*0*01	01*1*	1*0*1
01*1*	1*0*1	1**01

Tabela 3.4. Rezultat działania algorytmu *Q-Hidden* dla ciągu bitowego 10010 przy parametrach $k = 3$, $q = 0.5$ i $r = 4.2$

przez solwery SAT w czasie potęgowym tj. przez całkowite przeszukiwanie, co powoduje że dla odpowiednio dużych l odzyskanie zabezpieczonego ciągu jest niepraktyczne.

Takie podejście do problemu ma szereg konsekwencji. Pierwszą z nich jest to że wielkość otrzymanej bazy nie zależy od przechowywanego rekordu i można ją kontrolować za pomocą parametru r – generowane jest dokładnie $l * r$ rekordów. Nie ma jednak możliwości zawarcia w jednej instancji *NDB* więcej niż jednego ciągu bitowego - można zakodować jedynie zero lub jeden rekord.

W [7] razem z przedstawieniem algorytmu 7 opisany jest przykładowe zastosowanie tego schematu w systemach rzeczywistych wymagających znacznie większych ilości danych. Polega na przechowywaniu zbioru osobnych *NDB* dla każdego pozytywnego rekordu. Metoda zapytania do tak skonstruowanej bazy polega na sprawdzeniu po kolei wszystkich pojedynczych *NDB*. Jeżeli rekord nie pokrywa się z żadną formułą w jakiegokolwiek pod-bazie to przyjmuje się że znajduje się w bazie głównej.

Negatywne Bazy Danych nie przechowujące żadnych danych okazują się przydatne w takim rozwiązaniu ponieważ „puste” bazy stworzone przez algorytm *0-Hidden* (zmodyfikowany w celu generowania formuł o wymaganej liczbie literałów w klauzuli) są nierozróżnialne od tych wygenerowanych przez algorytm *Q-Hidden*. Pozwala to na ukrycie liczby rekordów pozytywnych.

Istotnym problemem jest fakt, że nie ma gwarancji, że rezultat działania tego algorytmu będzie pokrywał cały zbiór $U - \{s\}$, nawet powyżej wartości granicznej $r \approx 4.27$. Jedną z opcji jest dalsze zwiększenie parametru r żeby jeszcze bardziej zmniejszyć prawdopodobieństwo zawarcia nadmiarowych ciągów, ale wprowadza to dodatkowe, redundantne informacje co powoduje że czas potrzebny na odwrócenie *NDB* maleje. Dlatego wskazane jest ustawienie niskiej wartości r i dodanie do każdego rekordu dodatkowej informacji wskazującej na jego poprawność np. bitu parzystości, kodu CRC lub wyniku funkcji skrótu. W takiej sytuacji prawdopodobieństwo wystąpienia niechcianych danych maleje 2^c -krotnie, gdzie c oznacza liczbę dodatkowych bitów.

Rozkład wartości nadmiarowych ciągów bitowych nie jest jednostajny - dodatkowe wystąpienia będą bliskie s w odległości Hamminga (tj. poszczególne bity będą się różniły w niewielkiej ilości), dlatego

metoda CRC maksymalizująca odległość Hamminga dla podobnych napisów może być szczególnie efektywna. [7]

Porównanie poszczególnych metod przedstawiam w rozdziale 4.2.

3.3.6. Algorytm *K-Hidden*

K-Hidden jest modyfikacją algorytmu *Q-Hidden*. Zamiast pojedynczego parametru prawdopodobieństwa q stosuje się w nim wektor parametrów $\{p_1 \dots p_k\}$ w celu zwiększenia ilości stopni swobody algorytmu [8]. W ten sposób możemy dowolnie kontrolować dystrybucję różnych typów klauzul - w zależności od liczby pokrywających się bitów z ciągiem wejściowym.

Algorytm 8: Algorytm *K-Hidden*

Dane: s - ciąg bitowy, l - długość rekordu, r - wsp. ilości klauzul

k - ilość ustalonych bitów w klauzuli, $\{p_1 \dots p_k\}$ - wsp. prawdopodobieństwa

Wynik: Zbiór rekordów NDB

```

1  $n = l * r$ 
2  $NDB = \{\}$ 
3  $Q = \{Q_0, Q_1 \dots Q_k\} : Q_0 = 0, Q_i = p_1 + \dots + p_i$ 
4 while  $|NDB| \neq n$  do
5    $rnd = \text{losowa liczba rzeczywista z przedziału } (0, 1)$ 
6    $i = i : Q_{i-1} \leq rnd \leq Q_i$ 
7    $V = \text{rekord z } k \text{ ustalonymi bitami, pokrywający się z } s \text{ na } k - i \text{ losowych pozycjach}$ 
8   Dodaj rekord  $V$  do  $NDB$ 
9 end
```

Przed generacją każdego kolejnego rekordu losuje się liczbę z przedziału $(0, 1)$ i na podstawie jej i wektora parametrów p_i wybiera się odpowiedni typ rekordu - mający dokładnie i pokrywających się zmiennych gdzie i jest indeksem najmniejszego parametru większego od wylosowanej liczby.

Rezultat działania ma podobne cechy co wygenerowany przez *Q-Hidden* - możliwość zakodowania tylko jednego napisu i brak gwarancji pokrycia całego zbioru $U - \{s\}$.

W literaturze pojawia się także zaproponowany wcześniej algorytm *p-Hidden* będący szczególnym przypadkiem *K-Hidden* dla $k = 3$ [9]. Analogicznie dla przypadku gdy wektor parametrów p jest następujący:

$$p_i = \frac{\binom{k}{i} q^i}{(1 + q)^k - 1}, i = 1 \dots k - 1$$

algorytm jest równoważny *Q-Hidden*. [8]

3.3.7. Algorytm hybrydowy

Algorytm hybrydowy jest połączeniem algorytmu prefikсового oraz *Q-Hidden* [10]. Ma na celu zapewnienie, że wygenerowana baza będzie zawierać całe uniwersum napisów bez jednego, ukrytego rekordu i jednocześnie będzie odporna na odwrócenie. Baza wynikowa jest równoważna formule 3-CNF.

Algorytm 9: Algorytm hybrydowy

Dane: s - ciąg bitowy, l - długość rekordu, r - wsp. ilości klauzul

q - wsp. prawdopodobieństwa

Wynik: Zbiór rekordów NDB

1 $NDB = \text{GenComplete}(s, l)$

2 $NDB = \text{MakeHardReverse}(NDB, s, l, r, q)$

Proces generacji składa się z dwóch etapów. Pierwszy, procedura *GenComplete*, tworzy kompletny zbiór $l + 4$ rekordów pokrywający całe $U - \{s\}$. Na początku tworzy $l - 2$ rekordy kolejno zaprzeczając każdemu bitowi oprócz dwóch ostatnich i losuje dwie dodatkowe pozycje przed nim ustawiając je zgodnie z s . Następnie generuje dwa rekordy dla różniącej się pozycji drugiej i cztery dla pierwszej.

NDB jest tworzona dla losowej permutacji napisu s a następnie każdy rekord jest przekształcany z powrotem przez permutację odwrotną w celu utrudnienia analizy przez pomieszenie kolejności bitów.

W kolejnym etapie tworzone są pozostałe $l * r - (l + 4)$ rekordy przez zastosowanie procedury *MakeHardReverse* (Algorytm 11) - równoznacznej *Q-Hidden* dla $k = 3$.

Wynik działania tej metody dla nieznanego napisu s jest nierozróżnialny od baz utworzonych przez algorytmy *0-Hidden*, *1-Hidden*, *2-Hidden*, *Q-Hidden* i *K-Hidden* przy $k = 3$ i identycznym r . Różnić się będą jedynie rozkładem pozycji ustalonych zgodnych i niezgodnych z ukrytym napisem.

11000
*1*01
1*01*
111**
100**
101**
0**00
0**10
0**01
0**11

Tabela 3.5. Rezultat działania procedury *GenComplete* dla $\pi(s) = 11000$ przed zastosowaniem permutacji odwrotnej na rekordach

Algorytm 10: GenComplete**Dane:** s - ciąg bitowy, l - długość rekordu**Wynik:** Zbiór rekordów NDB

```

1   $\pi$  = losowa permutacja o długości  $l$ 
2  for  $k = l$  to 3 do
3       $V$  = rekord NDB o długości  $l$  wypełniony symbolami '*'
4       $V[k] = \neg\pi(s)[k]$ 
5       $i, j$  = dwie różne liczby z przedziału  $[1, k]$ 
6       $V[i] = \pi(s)[i]$ 
7       $V[j] = \pi(s)[j]$ 
8      Dodaj rekord  $V$  do NDB
9  end
10  $V$  = rekord NDB o długości  $l$  wypełniony symbolami '*'
11  $V[1] = \pi(s)[1]$ 
12  $V[2] = \neg\pi(s)[2]$ 
13  $i$  = losowa liczba z przedziału  $[3, l]$ 
14  $V[i] = 0$ , Dodaj rekord  $V$  do NDB
15  $V[i] = 1$ , Dodaj rekord  $V$  do NDB
16  $i$  = losowa liczba z przedziału  $[2, l]$ 
17 for  $b = 0$  to 1 do
18      $V$  = rekord NDB o długości  $l$  wypełniony symbolami '*'
19      $V[1] = \neg\pi(s)[1]$ 
20      $j$  = losowa liczba z przedziału  $[2, l]$ , różna od  $i$ 
21      $V[i] = b$ 
22      $V[j] = 0$ , Dodaj rekord  $V$  do NDB
23      $V[j] = 1$ , Dodaj rekord  $V$  do NDB
24 end
25 Zwróć  $\pi^{-1}(NDB)$ 

```

Algorytm 11: MakeHardReverse**Dane:** s - ciąg bitowy, l - długość rekordu, r - wsp. ilości klauzul q - wsp. prawdopodobieństwa**Wynik:** Zbiór rekordów NDB

```

1   $n = l * r$ 
2  while  $|NDB| \neq n$  do
3       $\Upsilon = 3$  różne losowe pozycje w przedziale  $[1, l]$ 
4       $V =$  rekord  $NDB$  o długości  $l$  wypełniony znakami '*'
5       $X =$  losowe przypisanie  $\in \{0, 1\}^3$ 
6       $u = 3$ 
7      foreach  $i = 1$  to 3 do
8           $V[\Upsilon[i]] = X[i]$ 
9          if  $V[\Upsilon[i]] \neq s[\Upsilon[i]]$  then
10              $u = u + 1$ 
11         end
12     end
13     if  $u \neq 0$  then
14          $p =$  losowa liczba rzeczywista z przedziału  $(0, 1)$ 
15         if  $q^u > p$  then
16             Dodaj rekord  $V$  do  $NDB$ 
17         end
18     end
19     Dodaj rekord  $V$  do  $NDB$ 
20 end
21 Zwróć powstałą  $NDB$ 

```

3.4. Wykorzystywanie solwerów SAT w celu uzyskania przeciwobrazu Negatywnej Bazy Danych

Problem znalezienia ukrytego ciągu w Negatywnej Bazie Danych jest równoważny SAT i zapis napisów nad alfabetem $\{0, 1, *\}$ można sprowadzić do CNF w czasie liniowym. Dla przykładu, poniższy zbiór rekordów można zinterpretować jako „Znajdź takie przypisanie wartościami $\{0, 1\}$, żeby dla każdego rekordu nie pokrywało się na co najmniej jednej pozycji”.

Zatem pomijając symbole $*$, zamieniając wartości $\{0, 1\}$ na odpowiadające literały, negując je i łącząc operatorem alternatywy oraz wstawiając operatory koniunkcji pomiędzy powstałe klauzule otrzymujemy postać CNF:

11*0
001*
*111
*101

Tabela 3.6. Przykładowy zbiór rekordów *NDB*

$$(\neg x_1 \vee \neg x_2 \vee x_4) \wedge$$

$$(x_1 \vee x_2 \vee \neg x_3) \wedge$$

$$(\neg x_2 \vee \neg x_3 \vee \neg x_4) \wedge$$

$$(\neg x_2 \vee x_3 \vee \neg x_4)$$

Metoda działania solwerów SAT nie polega na znalezieniu wszystkich rozwiązań, tylko na sprawdzeniu czy dana formuła jest spełnialna, dlatego w sytuacji próby odwrócenia baz wielorekordowych lub wygenerowanych z użyciem algorytmów niekompletnych atakujący potrzebuje dla każdego pojedynczego rozwiązania dodać dodatkową formułę przeczącą znalezionemu rekordowi i ponownie uruchomić solwer.

W związku z faktem, że uzyskanie dostępu do nawet pojedynczego rekordu w systemach uwierzytelniania jest niedopuszczalne, testy są przeprowadzone pod kątem znalezienia pierwszego rozwiązania.

4. Testy algorytmów

4.1. Opis testowania za pomocą solwerów SAT

4.2. Testy metod weryfikujących poprawność rekordów pozytywnych

4.3. Testy algorytmów generacji *NDB*

5. Implementacja systemu uwierzytelniania

5.1. Reprezentacja danych

5.2. Algorytm tworzenia użytkownika

5.3. Algorytm uwierzytelniania użytkownika

5.4. Działanie aplikacji

6. Wnioski

6.1. Bezpieczeństwo przedstawionej implementacji

6.2. Możliwe rozszerzenia

Bibliografia

- [1] Google / Harris Poll. *The United States of P@sswOrd\$*. <https://storage.googleapis.com/gweb-uniblog-publish-prod/documents/PasswordCheckup-HarrisPoll-InfographicFINAL.pdf>. 2019.
- [2] W. Luo i in. „Three Branches of Negative Representation of Information: A Survey”. W: *IEEE Transactions on Emerging Topics in Computational Intelligence* 2.6 (2018), s. 411–425. DOI: 10.1109/TETCI.2018.2829907.
- [3] Fernando Esponda. “Negative Representations of Information”. PhD thesis. 2005.
- [4] F. Esponda, S. Forrest i P. Helman. „Enhancing Privacy through Negative Representations of Data”. W: 2004.
- [5] Haixia Jia, Cristopher Moore i Doug Strain. „Generating Hard Satisfiable Formulas by Hiding Solutions Deceptively”. W: *Journal of Artificial Intelligence Research - JAIR* 28 (mar. 2005). DOI: 10.1613/jair.2039.
- [6] Dimitris Achlioptas, Haixia Jia i Cristopher Moore. „Hiding Satisfying Assignments: Two are Better than One”. W: *Proceedings of the National Conference on Artificial Intelligence* 24 (kw. 2005). DOI: 10.1613/jair.1681.
- [7] Fernando Esponda i in. „Protecting Data Privacy Through Hard-to-Reverse Negative Databases”. W: *Information Security*. Red. Sokratis K. Katsikas i in. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. ISBN: 978-3-540-38343-7.
- [8] D. Zhao i in. „A fine-grained algorithm for generating hard-to-reverse negative databases”. W: *2015 International Workshop on Artificial Immune Systems (AIS)*. 2015, s. 1–8. DOI: 10.1109/AISW.2015.7469244.
- [9] R. Liu, W. Luo i L. Yue. „The p-hidden algorithm: Hiding single databases more deeply”. W: *Immune Computation* 2 (sty. 2014), s. 43–55.
- [10] R. Liu, W. Luo i X. Wang. „A Hybrid of the prefix algorithm and the q-hidden algorithm for generating single negative databases”. W: *2011 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*. 2011, s. 31–38. DOI: 10.1109/CICYBS.2011.5949400.

- [11] Stephen A. Cook. „The Complexity of Theorem-Proving Procedures”. W: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. STOC '71. Shaker Heights, Ohio, USA: Association for Computing Machinery, 1971, 151–158. ISBN: 9781450374644. DOI: 10.1145/800157.805047.
- [12] B. A. Trakhtenbrot. „A Survey of Russian Approaches to Perebor (Brute-Force Searches) Algorithms”. W: *Annals of the History of Computing* 6.4 (1984), s. 384–400. DOI: 10.1109/MAHC.1984.10036.
- [13] Thomas J. Schaefer. „The Complexity of Satisfiability Problems”. W: *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*. STOC '78. San Diego, California, USA: Association for Computing Machinery, 1978, 216–226. ISBN: 9781450374378. DOI: 10.1145/800133.804350.
- [14] Adnan Darwiche i Knot Pipatsrisawat. „Complete Algorithms”. W: *Handbook of Satisfiability*. Red. Armin Biere i in. T. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, s. 99–130. DOI: 10.3233/978-1-58603-929-5-99.
- [15] Henry A. Kautz, Ashish Sabharwal i Bart Selman. „Incomplete Algorithms”. W: *Handbook of Satisfiability*. Red. Armin Biere i in. T. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, s. 185–203. DOI: 10.3233/978-1-58603-929-5-185.